Universal Content Launcher

A series of tutorials detailing the creation of a content launcher in Microsoft Visual C#

Advanced Tutorial Two

Steam Integration

Part B

Using Steam Content

Contents

Introduction **3**
Tutorial Requirements 3
Part One: Storing Steam data **4**
Part Two: Launching the Steam content **16**

Introduction

In our series of basic tutorials, we embarked on a journey to create a basic content launcher in C#. We've also looked briefly at the Windows API, and how we can call external functions from our own code.

In the first part of our Steam integration tutorial we looked at using a Steam emulator to return information from a clientregistry.blob file. Now we're going to look into using that information to launch our Steam content.

Tutorial Requirements

The advanced tutorials are enhancements to the first five basic tutorials that built our Universal Content Launcher. You can either work through these tutorials first, or download the source code, and have a go at this advanced tutorial. Advanced tutorials that are split into further parts should be treated as a single long tutorial that requires all of the parts to be understood correctly.

For the purposes of this tutorial, it is assumed that you fully understand everything that is in the downloadable source code.

Part One: Storing Steam data

In our first Steam advanced tutorial, we took a look at how we could retrieve information from the clientregistry.blob file. Now we need to store this information within UCL for when we finally need to use it.

We already have a class for storing content in UCL, so we could add any specific Steam related stuff to that, but that would mean a lot of non-Steam content would be forced to hold Steam related fields, which is messy – to say the least.

Luckily we can rely on our old friend inheritance to help out.

Create a new class in UCL and call it UCLSteamContent.cs, you should have something like this:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace UCL
{
    class UCLSteamContent
    {
    }
}
```

Now we need to add in some namespaces that are used in UCLContent.cs and set our new Steam class to inherit from the original UCLContent.cs class. What this means is that all of the types and methods available in the UCLContent class will now be available in the UCLSteamContent class also, without us having to write any extra code!

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Windows.Forms;

namespace UCL
{
    public class UCLSteamContent : UCLContent
    {
    }
}
```

Now that was easy!

Now let's modify our original Steam content code that is currently outputting information to a text file, to populate the information into our new Steam content class.

Do you remember way back when we defined an ArrayList to hold our content records in memory? I told you that an ArrayList can hold an array of any type, so it is quite possible to have both UCLContent and UCLSteamContent records in the same list.

Let's change the Type enum in UCLContent to help us Identify Steam content and then create a new enum in UCLSteamContent to hold the specific type of Steam content, like so:

In the UCLContent.cs class file I changed the enum like so:

```
public enum UCLType
{
    Game = 0,
    Application = 1,
    Video = 2,
    Audio = 3,
    Steam = 4,
}
```

In the UCLSteamContent.cs class file I added a new enum:

```
public enum UCLSteamType
{
    HalfLife = 0,
    HalfLifeMod = 1,
    Source = 2,
    SourceMod = 3,
    ThirdParty = 4,
    Tool = 5,
    SteamMedia = 6,
}
```

All this is doing, is breaking up the Steam content into what I think will be useful groups at a later stage, we can then add to this functionality later if required.

Now all of that is done, let's modify our code to store the Steam content in our new UCLSteamContent class and populate the tree view as we go.

First we need to add localised variables to our UCLSteamContent class to enable us to read and write the Steam specific values correctly, exactly like we did when we wrote the UCLContent class, so here's what I've got for UCLSteamContent.cs at the moment:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Windows.Forms;

namespace UCL
{
    public class UCLSteamContent : UCLContent
    {

        public enum UCLSteamType
        {
            HalfLife = 0,
            HalfLifeMod = 1,
            Source = 2,
            SourceMod = 3,
            ThirdParty = 4,
            Tool = 5,
            SteamMedia = 6,
        }

        UCLSteamType mSteamType;
        uint mAppID;
        uint mIsAppSubscribed;
        string mGCFList;

        public UCLSteamType SteamType
        {
            get
            {
                return this.mSteamType;
            }
            set
            {
                this.mSteamType = value;
            }
        }

        public uint AppID
        {
            get
            {
                return this.mAppID;
            }
            set
            {
                this.mAppID = value;
            }
        }

        public uint IsAppSubscribed
        {
            get
            {
                return this.mIsAppSubscribed;
            }
```

```
            set
            {
                this.mIsAppSubscribed = value;
            }
        }

        public string GCFList
        {
            get
            {
                return this.mGCFList;
            }
            set
            {
                this.mGCFList = value;
            }
        }

    }
}
```

That should all make perfect sense to you! Next we can modify our `LoadSteamContentList` method in UCLMain.cs to populate both the ArrayList and the Tree view control, like so; Note that I have identified the new lines by adding comments after them, and that the file output code has now been commented out. We will however leave it in there as it could come in handy at a later date:

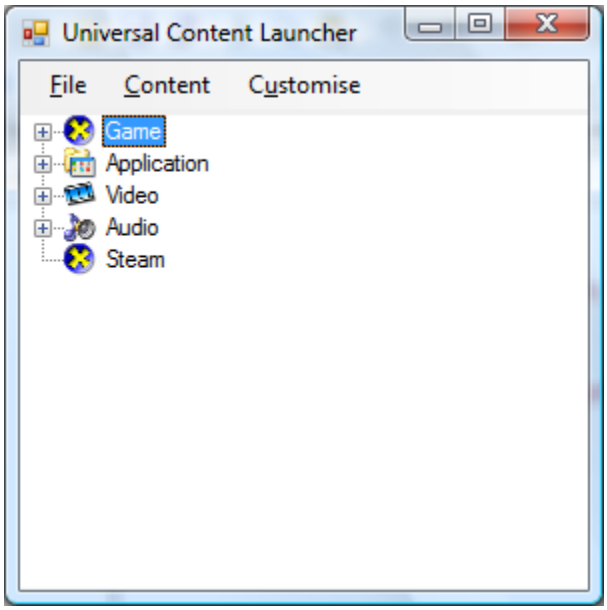
```
       public void LoadSteamContentList()
        {

            UCLSteamEmu.TSteamError SErr = new
UCLSteamEmu.TSteamError();
            uint iUserSet = 0;
            int iSteamReturn = 0;

            iSteamReturn = UCLSteamEmu.SteamSetUser("UCL", ref
iUserSet, ref SErr);

            if (iUserSet == 1)
            {
                UCLSteamEmu.TSteamAppStats AppStats = new
UCLSteamEmu.TSteamAppStats();

                iSteamReturn = UCLSteamEmu.SteamGetAppStats(ref
AppStats, ref SErr);

                if (AppStats.NumApps > 0)
                {

                    TreeNode tNode; //Create a new treeview node
                    tNode = treeView1.Nodes.Add("Steam"); //call it
Steam!


                    //FileInfo LogFile = new FileInfo("SteamApps.txt");
```

```
                    //StreamWriter TextLogFile = LogFile.CreateText();

                    uint []RawAppIDs;
                    RawAppIDs = new uint[AppStats.NumApps];
                    iSteamReturn =
UCLSteamEmu.SteamGetAppIds(RawAppIDs, AppStats.NumApps, ref SErr);

                    uint iBuffSize = 512;

                    foreach (uint RawAppID in RawAppIDs)
                    {

                        uint iIsAppSubscribed = 0;
                        uint iIsAppReady = 0;
                        iSteamReturn =
UCLSteamEmu.SteamIsAppSubscribed(RawAppID, ref iIsAppSubscribed, ref
iIsAppReady, ref SErr);

                        UCLSteamEmu.TSteamApp AppNFO = new
UCLSteamEmu.TSteamApp();

                        AppNFO.Name = new String(' ', 255);
                        AppNFO.maxNameChars = 255;
                        AppNFO.currentVersionLabel = new String(' ',
255);
                        AppNFO.maxCurrentVersionLabelChars = 255;
                        AppNFO.latestVersionLabel = new String(' ',
255);
                        AppNFO.maxLatestVersionLabelChars = 255;
                        AppNFO.cacheFile = new String(' ', 255);
                        AppNFO.maxCacheFileChars = 255;

                        iSteamReturn =
UCLSteamEmu.SteamEnumerateApp(RawAppID, ref AppNFO, ref SErr);

                        if (AppNFO.numDependencies != 0)
                        {

                            UCLSteamEmu.TSteamAppLaunchOption AppLO =
new UCLSteamEmu.TSteamAppLaunchOption();
                            AppLO.description = new String(' ', 255);
                            AppLO.maxDescChars = 255;
                            AppLO.cmdLine = new String(' ', 255);
                            AppLO.maxCmdLineChars = 255;
                            AppLO.index = 0;
                            AppLO.iconIndex = 0;
                            AppLO.noDesktopShortcut = 0;
                            AppLO.noStartMenuShortcut = 0;
                            AppLO.isLongRunningUnattended = 0;

                            iSteamReturn =
UCLSteamEmu.SteamEnumerateAppLaunchOption(RawAppID, 0, ref AppLO, ref
SErr);

                            StringBuilder strAppDir = new
StringBuilder((int)iBuffSize);
```

```
                            iSteamReturn =
UCLSteamEmu.SteamGetAppDir(RawAppID, strAppDir, ref SErr);

                            StringBuilder GCFList = new
StringBuilder((int)iBuffSize);
                            iSteamReturn =
UCLSteamEmu.SteamEmu_GetGcfList(RawAppID, GCFList, iBuffSize);


                            UCLSteamContent sContent = new
UCLSteamContent(); //Create a new UCLSteamContent record
                            TreeNode newNode =
tNode.Nodes.Add(AppNFO.Name); //Create a treeview record by name for
the steam content under the Steam category

                            //TextLogFile.WriteLine("Steam AppID:
" + RawAppID);
                            //TextLogFile.WriteLine("      Subscribed:
" + iIsAppSubscribed);
                            //TextLogFile.WriteLine("      Name:
" + AppNFO.Name);
                            //TextLogFile.WriteLine("      Path:
" + strAppDir.ToString());
                            //TextLogFile.WriteLine("      cmdLine:
" + AppLO.cmdLine);
                            //TextLogFile.WriteLine("      GCFList:
" + GCFList.ToString());
                            //TextLogFile.Write(TextLogFile.NewLine);

                            // The lines below populate the
                        UCLSteamContent record, and add it to the
                        UCLContentList ArrayList
                            sContent.Type = UCLContent.UCLType.Steam;
                            sContent.AppID = RawAppID;
                            sContent.IsAppSubscribed =
iIsAppSubscribed;
                            sContent.IsAppReady = iIsAppReady;
                            sContent.State = statebuff.ToString();
                            sContent.Name = AppNFO.Name;
                            sContent.ExePath = strAppDir.ToString();
                            sContent.ExeName = AppLO.cmdLine;
                            sContent.GCFList = GCFList.ToString();

                            UCLContentList.Add(sContent);
                            sContent = null;

                        }

                    }

                    //TextLogFile.Close() ;

                }
                else
                {
```

```
                    MessageBox.Show("Could not find any Steam content
to load!");
                }

            }else{

                MessageBox.Show("Could not set the Steam User!");

            }

        }
```

Ok, let's run UCL and see what happens:



Hmm, some things to Note:

- The Steam Category has no Icon – you should be able to fix that one yourself, refer to Advanced Tutorial One if you are having trouble remembering how we assigned icons to categories.

- The Steam Category has no content. Why? Well look at this:

```
private void UCLMain_Load(object sender, EventArgs e)
{
    LoadSteamContentList();

    LoadContentList();

}
```

We do the `LoadSteamContentList()` method before the `LoadContentList()` method. Then if we look at the code in the `LoadContentList()` method, it contains this line:

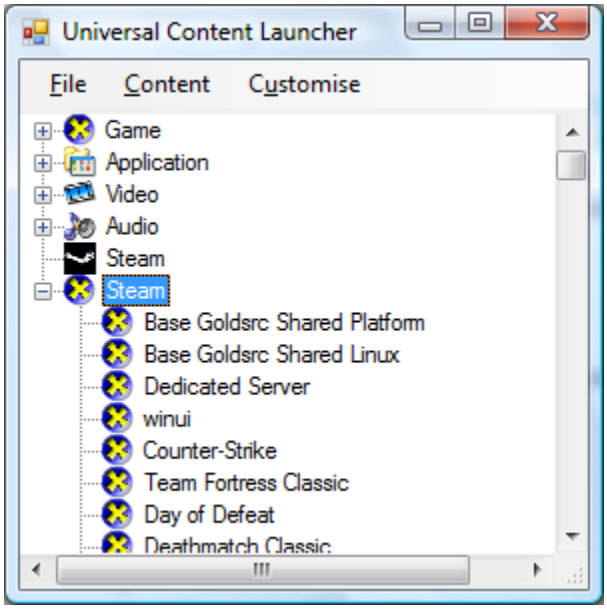
```
treeView1.Nodes.Clear();
```

So we did all the work in `LoadSteamContentList()`, populated the Tree view and then cleared it in the next method! How good are we?

Well, we could sort that by simply swapping the order of the method calls in `UCLMain_Load()`. So let's do that so it reads like this:

```
private void UCLMain_Load(object sender, EventArgs e)
{

    LoadContentList();
    LoadSteamContentList();

}
```

And try it again:



Ok, we can deduce a few things from the screenshot above.

First, the Steam category is processed twice, once for our normal content loop, that processes all the types in the UCL Content Type enum, remember? And then it runs our Steam content loop, which we coded to add its own Steam node to the tree view.

A better option for us would be to call the Steam content loop at the appropriate time within our original loop in `LoadContentList()`. To do this we would have to

pass to the `LoadSteamContentList()` method the tree view node that we want to add the content to, so remove the `LoadSteamContentList()` method from `UCLMain_Load()`:

```
        private void UCLMain_Load(object sender, EventArgs e)
        {

            LoadContentList();

        }
```

So we're back where we started! Now modify the `LoadSteamContentList()` method as follows:

```
        public void LoadSteamContentList(TreeNode SteamNode)
```

And remove from it the following lines:

```
                    TreeNode tNode; //Create a new treeview node
                    tNode = treeView1.Nodes.Add("Steam"); //call it
Steam!
```

And change the following line of code from:

```
                            TreeNode newNode =
tNode.Nodes.Add(AppNFO.Name);
```

To:

```
                            TreeNode newNode =
SteamNode.Nodes.Add(AppNFO.Name);
```

Now we need to modify the original `LoadContentList()` method to call our `LoadSteamContentList()` method at the appropriate time, here is an excerpt from that method with a new switch statement in place to do the job:

```
                    switch (tType){
                        case "Steam":
                            LoadSteamContentList(tNode);
                            break;
                        default:
                            XPathExpression expr;
                            expr = nav.Compile("/Content/" + tType +
"/Name");
                            XPathNodeIterator iterator =
nav.Select(expr);

                            try
                            {
                                while (iterator.MoveNext())
                                {
```

```
                                    UCLContent tContent = new
UCLContent();

                                    XPathNavigator nav2 =
iterator.Current.Clone();
                                    Type type = tContent.GetType();
                                    PropertyInfo tTemp =
type.GetProperty(nav2.Name);
                                    tTemp.SetValue(tContent,
nav2.Value, null);
                                    TreeNode newNode =
tNode.Nodes.Add(tContent.Name);

                                    tTemp = type.GetProperty("Type");
                                    UCLContent.UCLType tUCLType =
(UCLContent.UCLType)Enum.Parse(typeof(UCLContent.UCLType), tType,
true);
                                    tTemp.SetValue(tContent, tUCLType,
null);

                                    while (nav2.MoveToNext())
                                    {
                                        tTemp =
type.GetProperty(nav2.Name);
                                        tTemp.SetValue(tContent,
nav2.Value, null);
                                    }

                                    if
(File.Exists(Path.Combine(Application.StartupPath + "\\Icons\\",
tContent.Name + ".ico")))
                                    {
                                        tIcon = new
System.Drawing.Icon(Path.Combine(Application.StartupPath + "\\Icons\\",
tContent.Name + ".ico"));
                                        imageList1.Images.Add(tIcon);
                                        tNode.ImageIndex = tIconCount;
                                        tNode.SelectedImageIndex =
tIconCount;
                                        tIconCount++;
                                        tIcon = null;
                                    }
                                    else
                                    {

                                        tIcon =
UCL.Utility.GetAppIcon(Path.Combine(tContent.ExePath,
tContent.ExeName));
                                        imageList1.Images.Add(tIcon);
                                        newNode.ImageIndex =
tIconCount;
                                        newNode.SelectedImageIndex =
tIconCount;
                                        tIconCount++;
                                        tIcon = null;
                                    }
```

```
                                UCLContentList.Add(tContent);

                                tContent = null;
                            }
                        }
                        catch (Exception ex)
                        {
                            Console.WriteLine(ex.Message);
                            MessageBox.Show("Cannot load the UCL
content list. There is a problem processing the UCLContent.xml file!");
                        }
                        break;
                }
```

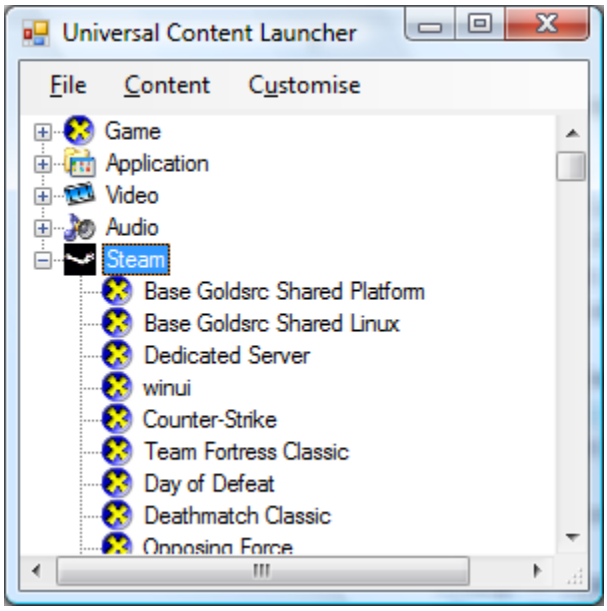
If you haven't figured it out yet, the only new lines added to this code are the first five lines:

```
                switch (tType){
                    case "Steam":
                        LoadSteamContentList(tNode);
                        break;
                    default:
```

And the last two:

```
                        break;
                }
```

Now run UCL:



Ah, that's better – we're back on track now! And doing a switch statement in this manner means that if we ever have to support another type of content management system, which is what Steam is, then we just need to add a new

UCL Type, modify our switch statement and write a new method to load the new content!

So, now we need to see if we can actually launch anything, and we need to clear out any content that we don't actually have available, as it's pointless showing it in the list. Also, individual icons for all of that stuff would be nice. So let's see how much of that we can do in this tutorial!

Part Two: Launching the Steam content

To launch steam content, our launcher needs to mount all the GCF files correctly before running the associated executable with any possible launch options. Now this may seem like a horror story, but fortunately there is a function in Steam that does all this for us:

```
        [DllImport("steam.dll")]
        public static extern int SteamLaunchApp(uint AppId, uint opt,
string Args, ref TSteamError SteamErr);
```

`SteamLaunchApp` not only mounts the GCF files and launches the application, it also extracts what are known as the "minimum footprint" files if they are not already extracted. And another bonus on top of all that, is that hCUPa's SteamEmu will also compare the minimum footprint files to check if they are outdated to the GCF file, and re-extract them if they are, keeping your files up to date whenever you update the GCFs. Neat heh?

So, add the function prototype above to your UCLSteamEmu.cs class file.

Now to call it correctly, we need to do a couple of things:

- First, we need to write an override `Launch()` method for our UCLSteamContent class that will enable us to launch Steam content in a different way (via `SteamLaunchApp`) to the other content.

- Then, we need to check the return type of our `GetContentFromName()` method in UCLMain.cs to see if the content is Steam content. If it is, we need to cast the `UCLContent` variable to type `UCLSteamContent` to ensure the correct launch function is called.

So, in your UCLSteamContent.cs class file, add this method:

```
        public override Boolean Launch()
        {

            try
            {

                UCLSteamEmu.TSteamError SErr = new
UCLSteamEmu.TSteamError();
                int iSteamReturn = 0;
                iSteamReturn = UCLSteamEmu.SteamLaunchApp(this.AppID,
0, "", ref SErr);
                return true;

            }

            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return false;
            }

        }
```

Quite a simple method for a change! This simply calls `SteamLaunchApp` with the correct AppID to launch the content, and has a little bit of error reporting built in. We use the `override` keyword because the `Launch()` method also exists in the `UCLContent` class, which is of course the base class for `UCLSteamContent`.

Now in UCLMain.cs in our `treeView1_DoubleClick` method, we need to add a new switch statement as follows:

```
        private void treeView1_DoubleClick(object sender, EventArgs e)
        {

            UCLContent tContent =
GetContentFromName(treeView1.SelectedNode.Text);
            if (tContent != null)
            {
                switch (tContent.Type){
                    case UCLContent.UCLType.Steam:
                        UCLSteamContent tSteamContent =
(UCLSteamContent)tContent;
                        tSteamContent.Launch();
                        break;
                    default:
                        tContent.Launch();
                        break;
                }
            }
            else
            {
                MessageBox.Show("Cannot find the selected content to
launch!");
            }
        }
```

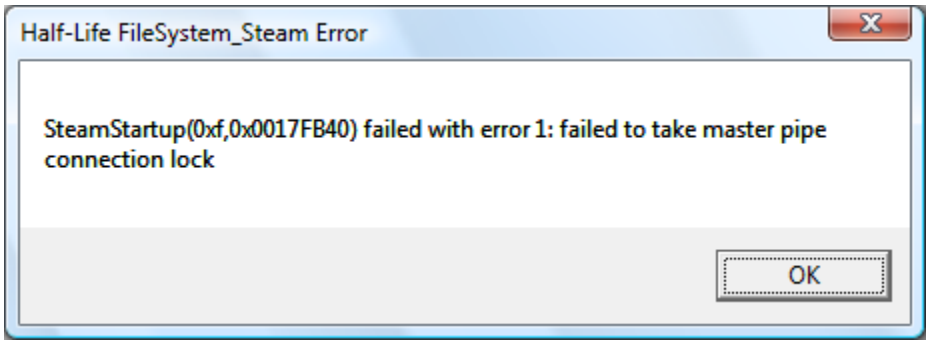
So, we're still calling `GetContentFromName()` in exactly the same way, and using that to determin if we have valid content. We can do this because all the objects in our content ArrayList do contain the class definition for the type `UCLContent`, so it is a valid type to use as a comparrison for all our content, whether it is "normal" contant or the extended "Steam" class content.

We then switch `tContent.Type` to determin if it is Steam content, and cast to type `UCLSteamContent` if it is, before calling the launch method.

Not try running something simple, like Half-Life. It is recommended that you have at least one type of each Steam content type to test on your launcher, otherwise somebody somewhere will no doubt find a bug for you!

Your results will vary largely depending on whether or not you have steam installed and it's registry keys exist to point steam to the correct folder to load some required dll files. Basically, some of you may see Half-Life up and running and some of you may see an error, if you don't have steam installed.

I got this one, though yours could be different:



We can fix any dll related errors by ensuring that the required dll's are available in our UCL program folder, rather than relying on the existence of a Steam installation. The files we need to run without error are as follows:

steam_api.dll
steamclient.dll
tier0_s.dll
vstdlib_s.dll

Along with the required Emulator files that we already have:

ClientRegistry.blob
steam.dll
steamemu.ini

If you need copies of these files, or are unsure of the versions of the files you should be using, then look at grabbing them from the UGL update pack at cs.rin.ru, which contains all of the files above.

If a user has Steam installed, then these files may or may not be a problem depending on their version in the steam folder. In order to gaurantee that our program has the maximum chance of working, then we really need to copy these files in to our UCL project to control which versions being used.

So copy them in to your UCL project folder, Include them in the project, and set them to Copy if Newer. Run UCL again and you should get exactly the same error (assuming you got one in the first place)

This is because we also need to modify a key in the registry, to tell SteamEmu where our “Steam” installation resides when it needs to access these dll files, i.e. our UCL folder which now has these files included.

The registry key in question is
HKEY_LOCAL_MACHINE\Software\Valve\Steam\InstallPath

You can look to see if it is there by running regedit (type regedit in the start box of your start menu and hit return) and browsing to the path above in the registry editor.

Note: If you are running a 64 bit version of windows, then you browse to the key like this:

HKEY_LOCAL_MACHINE\Software\Wow6432Node\Valve\Steam\InstallPath

It is worth pointing out though, that despite the extra level of Wow6432Node in 64 bit operating systems, you can read and write to it in exactly the same way you would do in any 32 bit windows installation, x64 handles the Wow6432Node for you automatically.

If the key exists then you can simply modify the value for InstallPath to your UCL debug folder, and you will see that Half-Life will now launch correctly. Remember to make a note of what the value was prior to modification, in case you need to change it back.

If the key does not exist then you can create it with a simply registry file, by pasting the following text into a new text file and calling it something like UCLSteamInstall.reg. Make sure the InstallPath points correctly to where your UCL project debug folder is, before running it.

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Valve\Steam]
"InstallPath"="C:\\Projects\\UCL\\bin\\Debug\\"
```

Note that as mentioned above, this file is valid for both 32 and 64 bit windows systems.

Needless to say, we have done this “quick” hack to check that the launching of Half-Life now works correctly, but we need to manage this automatically within our UCL program itself whenever it is run. So UCL would need to read this key and store its current value prior to changing it, whenever it is run, and restore the original value, when UCL is closed.

We’ll address this later, but for now, did it work?

Try runnning more content that you know you have all the available GCFs or NCFs + common folders for.

What you should notice, is that whenever you run a new game, a folder is created under your user folder in the directory that you defined in steamemu.ini as containing your GCF files.

Ok, at this point, most of my stuff is working! But we need to test some more, and our Steam content list is a mess, I mean it isn’t listed in any type of order, and also lists stuff that we can’t or wouldn’t want to run – like winui, or any Demos, or Betas. There also aren’t any icons for the Steam content, and we can even attempt to run stuff that we don’t even have all the files for (either GCF or NCF) resulting in various errors!

Well, we still have a lot of work to do! But to conclude this tutorial, let’s put in a simple check to determin if we have the files required to launch the selected Steam content. This is easy to do, because we have already stored the GCFlist required for the content to launch, we can test if all of the files exist first and show an appropriate error message if they don’t, by modifying our Steam content `Launch()` method like so.

Notice that I have made comments where appropriate:

```
        public override Boolean Launch()
        {

            try
            {
                string sSteamApps = "C:\\Games\\Steam\\SteamApps\\";
//the GCF file location should be pulled either from the UCL config
file or the SteamEmu.ini file, hard coding is bad, but for now in order
to show things working, it will do!
                int FailedFileCheck = 0;
                string GCFListings = "";
                string CurrentGCF = "";
                string[] SEGCFList =
this.GCFList.ToString().Split(";".ToCharArray());//this line and the
next splits our GCFList string into seperate strings of only 1 GCF file
each, so that we can then loop through them
                foreach (string sgcf in SEGCFList)
                {

                    if (sgcf.IndexOf("@") > -1) //an optional GCF
should not stop the content from running, but we still want to notify
the user of its existence if the launch fails
                    {
                        CurrentGCF = sgcf.Substring(1, sgcf.Length -
1);
                        CurrentGCF = CurrentGCF.ToLower() + "
(optional)";
                    }
                    else
                    {
                        CurrentGCF = sgcf;
                        if (!File.Exists(Path.Combine(sSteamApps,
CurrentGCF)))
                        {

                            if
(this.ExePath.ToLower().IndexOf("common") > -1) //we need to check
whether or not this is an NCF game as SteamEmu_GetGcfList always
contains a GCF file extension, even if the game is in NCF format! This
may or may not get fixed by hCUPa at a later date, but for now it's a
minor work around
                            {
                                if
(!File.Exists(Path.Combine(sSteamApps,
CurrentGCF.ToLower().Replace("gcf", "ncf"))))
                                {

                                    FailedFileCheck = 1;
                                    CurrentGCF =
CurrentGCF.ToLower().Replace("gcf", "ncf") + " + common folder";

                                }

                            } else {

                                FailedFileCheck = 1;
                                CurrentGCF = CurrentGCF.ToLower();
```

```
                            }
                        }
                    }

                    GCFListings = GCFListings + "\n" + CurrentGCF;
                }

                if (FailedFileCheck == 1)
                {
                    MessageBox.Show("You need the following content
files to run " + this.Name + "\n" + GCFListings); //we don't have all
the files, so show the user what they need - aren't we helpfull!
                    return false;
                }
                else
                {

                    UCLSteamEmu.TSteamError SErr = new
UCLSteamEmu.TSteamError();
                    int iSteamReturn = 0;
                    iSteamReturn =
UCLSteamEmu.SteamLaunchApp(this.AppID, 0, "", ref SErr);
                    return true;
                }
            }

            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return false;
            }

        }
```

Great, so that leaves us with the following things left to do, and no doubt there are more:

- We need to set and restore the Steam Installation path in the registry automatically when UCL is run and terminated.
- We need to read the value from steamemu.ini or the UCL config file for the GCF file location in our `Launch()` method, so it is not hard coded.
- We need to filter the Steam content list so only valid content (i.e. no demos, betas and system files) are shown.
- We need to add appropriate icons for the Steam content.
- We need to handle Half-Life and Source mods.
- We need to modify Steam content to enable launching with arguments.
- We could also impliment our Steam specific types to further break up the Steam Content List and make it more readable.

We had better get some rest, I think we're going to need it!